Report Number: WUCSE-2007-40

2007

A Fingerspelling Sign Language Visualization

Carol S. Brickman

The goal of the Fingerspell Visualization Project is to research methods to improve learning of reading skills through sign language. The techniques are centered on Fingerspelling as the method to bridge stages of skill development. Visualization of a string of text in images of a hand performing the letters of the alphabet in standardized fingerspell sign language positions provide Full Motion Learning as opposed to learning from single pictures.

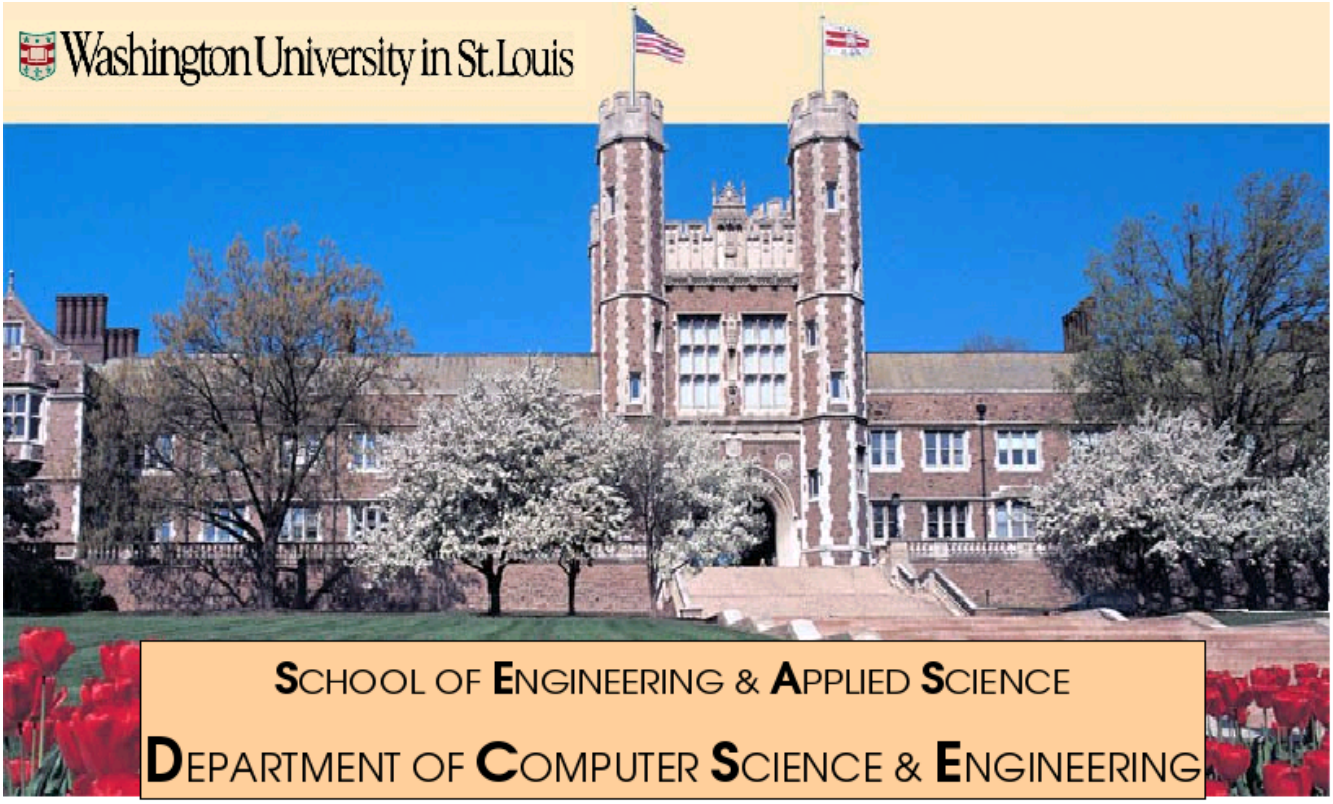
2007-40

A Fingerspelling Sign Language Visualization

Authors: Carol S. Brickman

Corresponding Author: csb1@cec.wustl.edu

Type of Report: Other



WUCSE-2007-40: A Fingerspelling Sign Language Visualization

Project Report by Masters Student: Carol S. Brickman

Advisor: Professor Cindy Grimm
Masters Project Committee:

Dr. Cindy Grimm
Dr. Christopher Gill
Dr. Tao Ju

April 24, 2007

Table of Contents

Provided on CD Included with Report:

A. Tar file of Fingerspell Database
B. Tar file of This Report, All software and Scripts
C. Mpegs used in Demo

Figures:

I. Fingerspell Objective

I.1 Introduction to Fingerspell Visualization

The goal of the Fingerspell Visualization Project is to research methods to improve learning of reading skills through sign language. The techniques are centered on Fingerspelling as the method to bridge stages of skill development. Visualization of a string of text in images of a hand performing the letters of the alphabet in standardized fingerspell sign language positions provide **Full Motion Learning** as opposed to learning from single pictures.

I.1.1 American Sign Language in Education

Several Sign Languages were developed as a communication tool for Hearing Impaired individuals, one of which is the "American Sign Language", (ASL). There have been many studies of the use of ASL in Early Childhood Learning. A list of a few of the studies is given in Appendix D. Teaching some ASL signs to Toddlers before they are able to speak has been integrated into the early childhood learning curriculum across the United States. Sign Language can be performed before the ability to speak avoiding the frustration of not being able to communicate needs. Some of the very many ASL signs for toddlers are: eat, drink, more, all gone, hot, apple, dog, cat, bird, book.[1] Furthermore, ASL is being used for children, older than toddler age, that are autistic or speech impaired. Public school districts provide adult education classes in ASL. ASL is used across the spectrum from children with disabilities to children for accelerated learning purposes.

I.1.2 Fingerspelling in Education

Not every word in the English Language is represented by an ASL sign. For such words, a hearing impaired individual will spell the word out in Fingerspell. The signs used in Fingerspelling have evolved and there are different versions. The Fingerspell signs used in this project are widely accepted for hearing impaired when using ASL.

I.1.3 Educational Community Uses

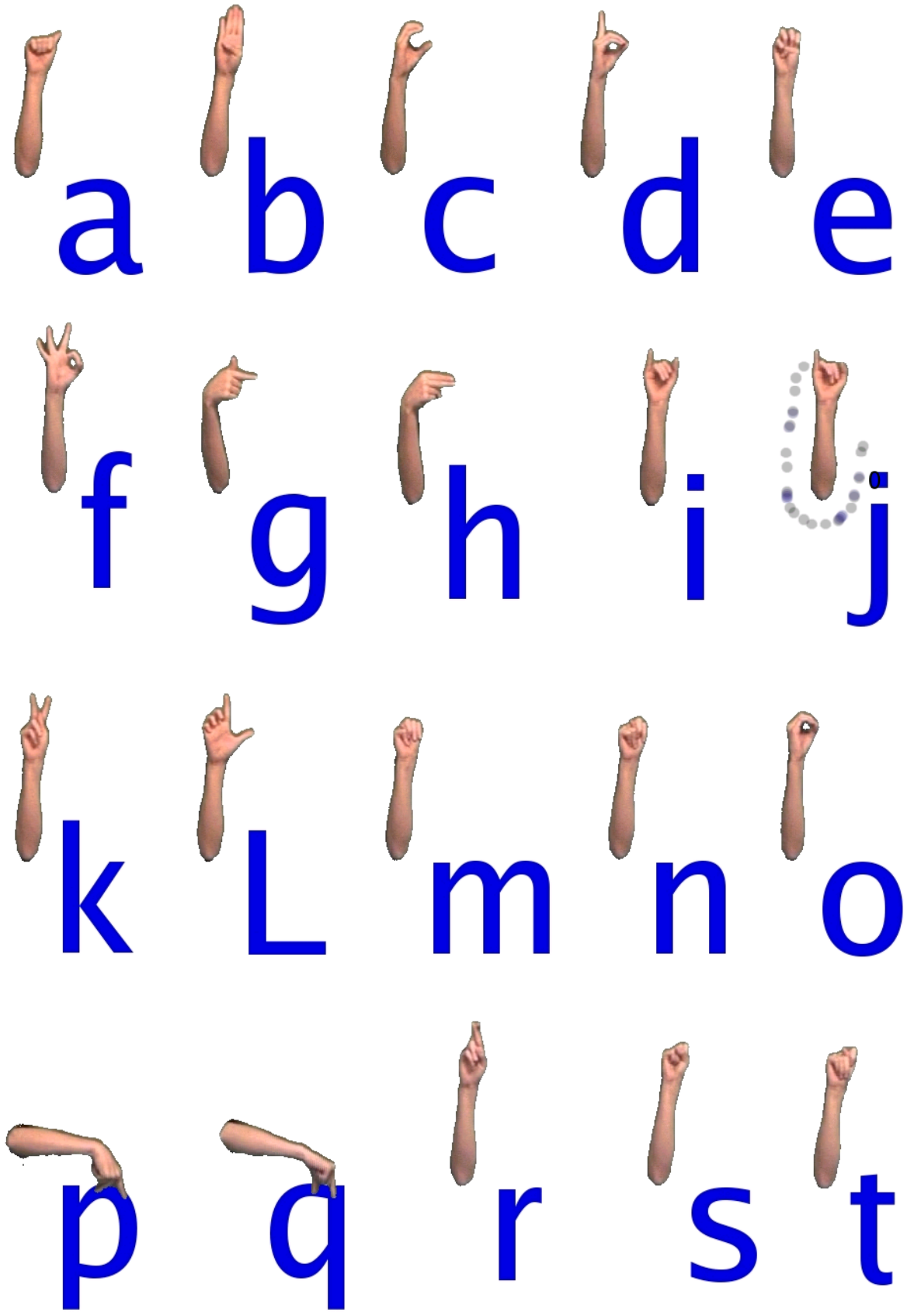
Products for ASL and Fingerspelling could be based on a collaborative area of study between Education, Psychology and Computer Science & Engineering. Computer labs to practice ASL and Fingerspelling would be of great use for both early childhood learning and adult learning. User friendly software products, available to classroom instructors, would put a tool into the hands of experts that allow them to create an activity personalized to the dynamics of their class and its educational needs.

I.2 Educational Software Development Examples

The following figures, Figure I.1-I.3, are three practical education Fingerspell uses, each for a different objective in early childhood education. Figures I.1 and I.2 are the static versions of the related movies which do the entire fingerspell for the expressed letter.

I.2.1 Learning the Alphabet

The Figure I.1 is geared toward a child that is learning the alphabet. Figure I.2 represents the learning stage where a child knows the ASL signs for a word, and is learning how to spell the word. The Figure I.3 is a game that practices the use of fingerspelling in learning to recognize a letter within a word, in this case it will always be the first letter.


a b c d e
f g h i j
k L m n o
p q r s t

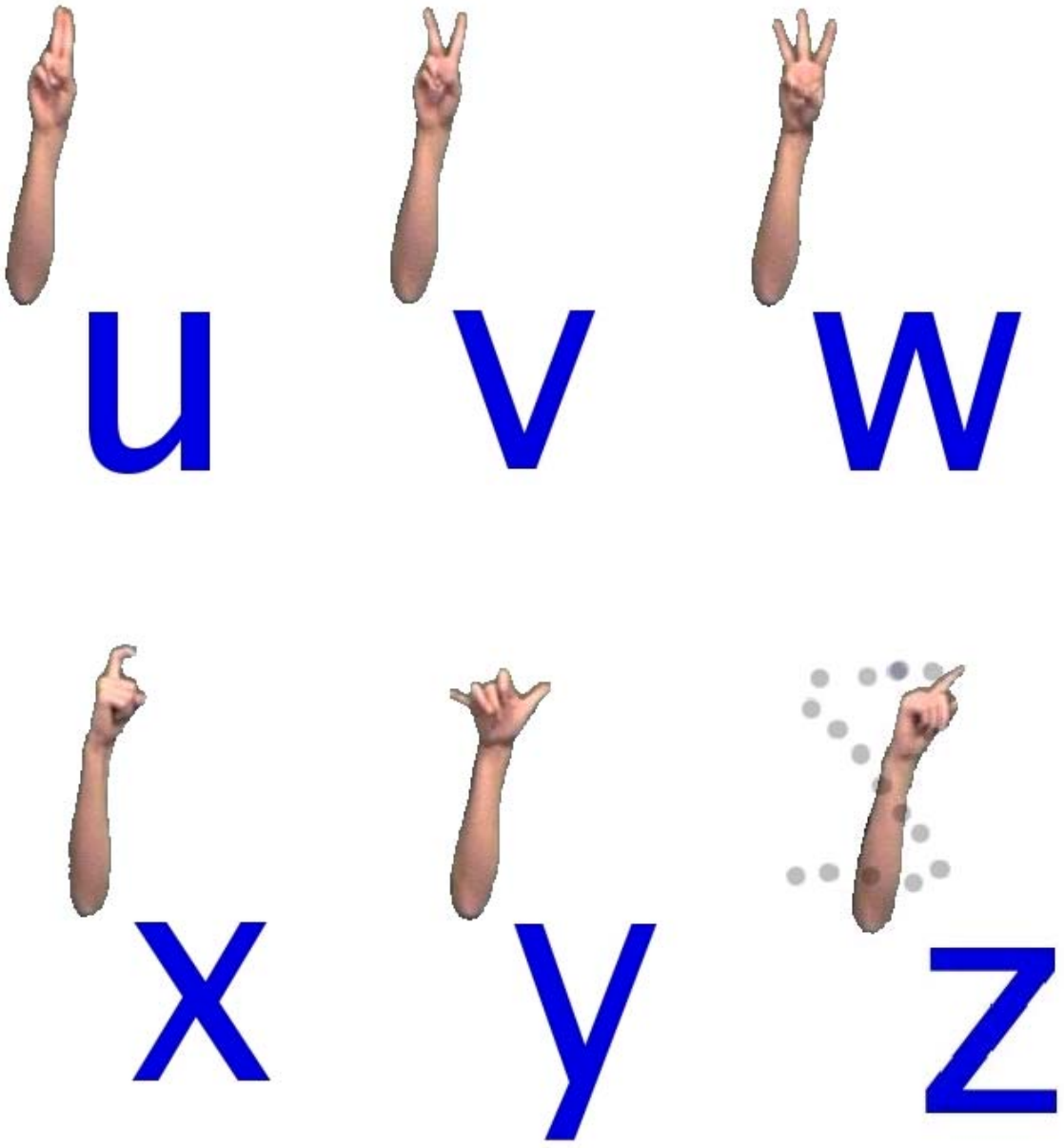


Figure I.1 Learning the Written Alphabet

In the video version of Figure I.1, the alphabet is spelled out, giving more meaning to the letters. Dots were used in Figure I.1 to indicate the movement required to form the letters "j" and "z". Each letter has over 20 jpegs encoded into the mpeg movie. The eye has difficulty discerning the finger position for the letter "m" based on a single picture. When presented as a movie, the hand is seen moving into position. Full motion makes it much easier to recognize the target position.

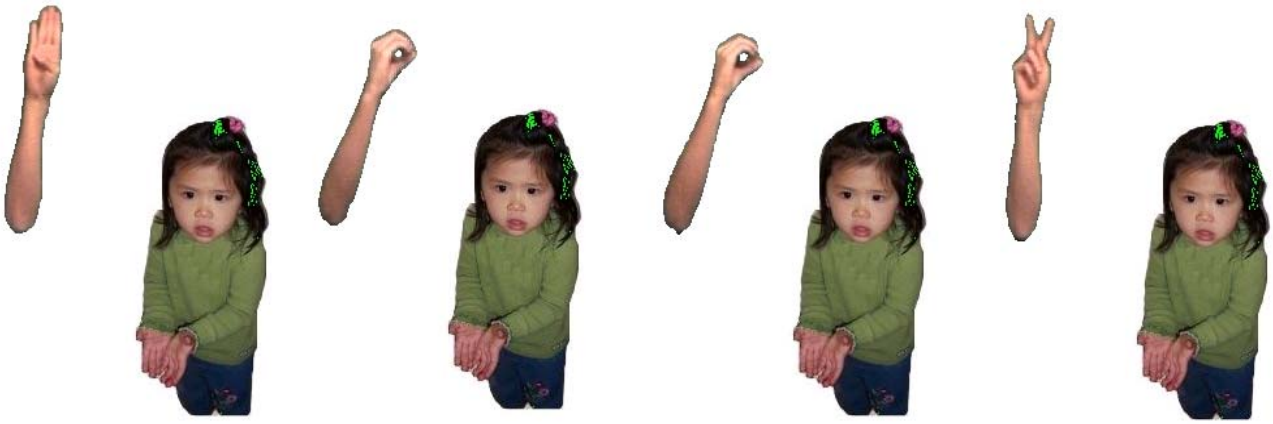


Figure I.2 Bridge from ASL to Spelling

I.2.2 Bridge from ASL to Spelling

In the video version of Figure I.2, The picture of the toddler does not change as the word "book" is spelled out and there are many frames in between the ones presented for this figure. A child that is already familiar with the ASL sign, could begin to recognize that there is an alternative method to communicate the same concept.

The word "book" was chosen for this presentation because it is a common word taught to toddlers and is an element in the University seal for Washington University in St. Louis. The book bears the University motto, *Per Veritatem Vis*, "Strength through Truth." Washington University was founded in 1853 as Eliot Seminary and renamed in 1857, recognized by an inscription on an archway in the Quadrangle, *Discere Si Cupias Intra: Salvere Lubermus*, "If you desire to learn, enter: We bid you welcome"

Figure I.3 Fingerspell Game

I.2.3 Fingerspell Game

Figure I.3 Is a pseudo screen capture of an application written for this project, fs2abc. The application spells out the entire word, without the underline. The game is the manner in which the teacher uses the application to practice fingerspelling. A classroom of kindergartners could be given a handout with the first letter missing. The teacher asks the question, "What letter is missing for the words, cup, cow and cat?". The students would silently hold up their hands in the fingerspelling of the letter "c". The teacher would then present the solution using a projection of the application onto a screen in the front of the classroom, by running an application such as, fs2abc. He/She would type in the word "cat", and the students would watch the fingerspelling of the word. The students could follow up by mimicking the fingerspelling of the entire word, though the focus is only on the first letter.

I.2.4 Fingerspell Visualization Application - fsvis and fs2abc

In order to allow a user to visualize a User Request String, an application was developed that reads and parses a text string, then displays the fingerspelling of the string. The speed of the display can be adjusted by the user. Figure I.4 shows the application developed to meet this objective.

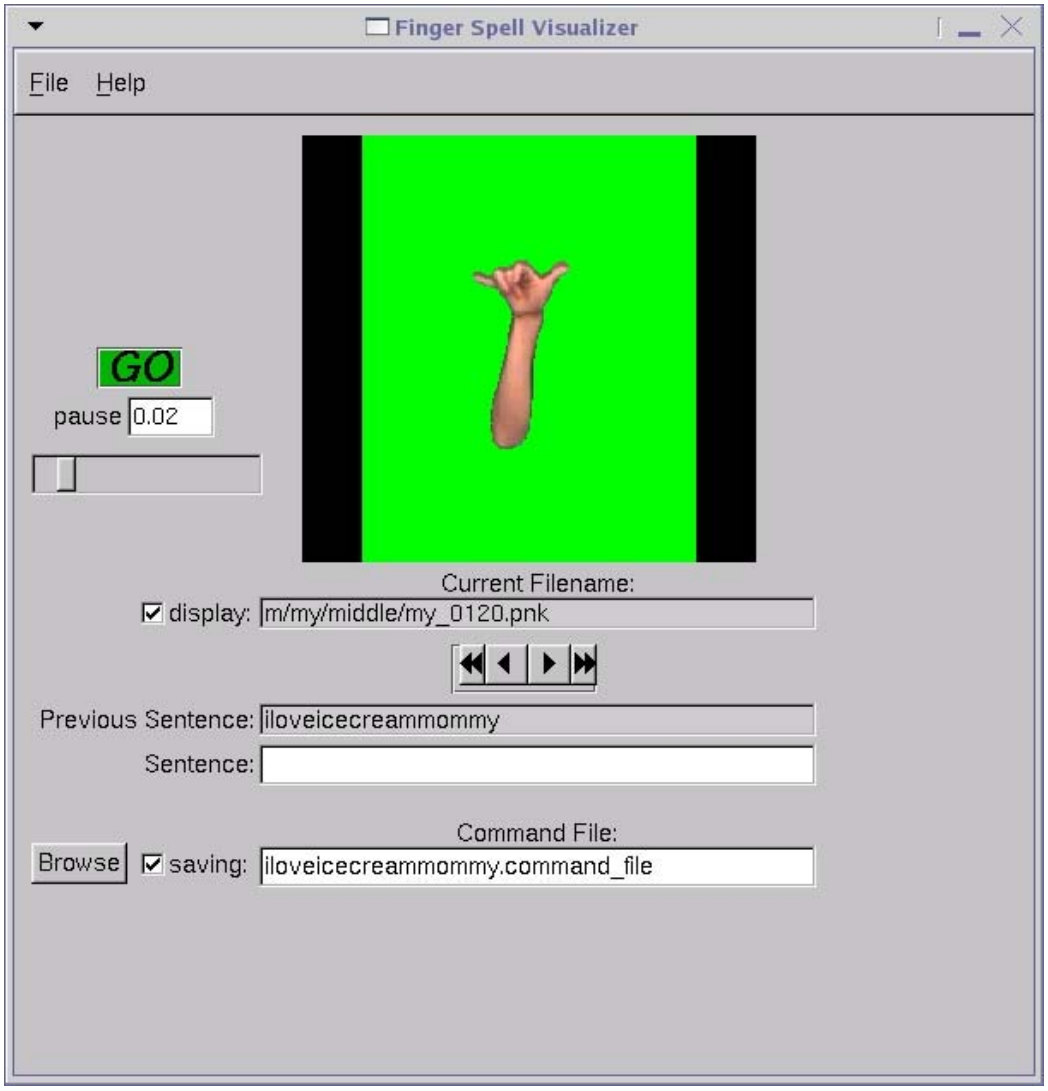


Figure I.4a fsvis application

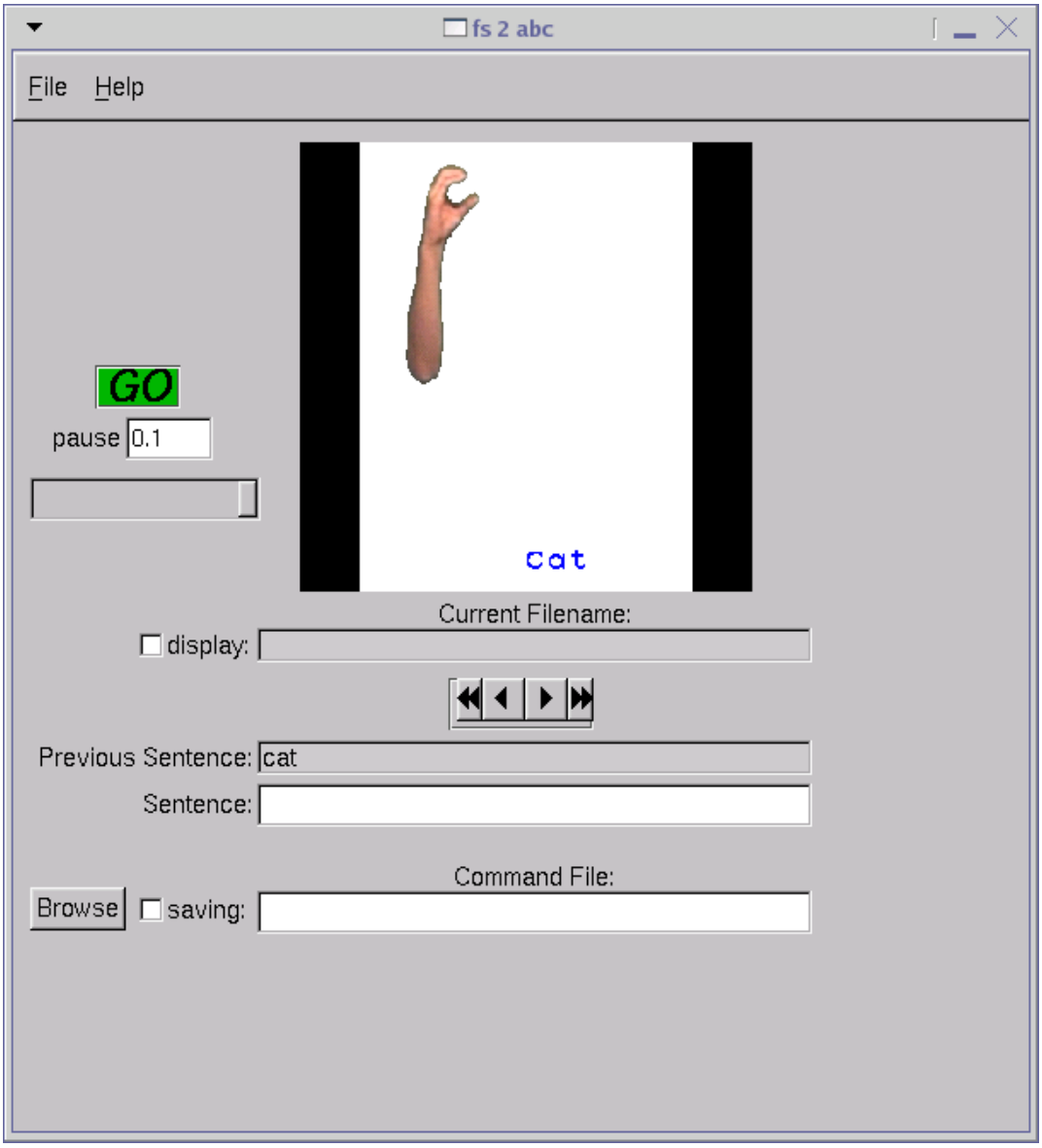


Figure I.4b fs2abc application

The Figure I.4b is a slightly different version of the fsvis program. In this version, the background is white, the hand is translated to a different position, and a single word is spelled out using a font provided by the application. The font, as is, is in a rough draft stage. The fingerspelling signs mimic the lower case letters. Likewise, the font used in a fingerspelling software should resemble the fingerspelling sign. The letter, "*l*", is a particular problem, because the fingerspelling sign looks more like a capital "L". A font specialized for use with fingerspelling would be necessary. Again, the current fs2abc application font is meant to emphasize the idea, not present the

optimized solution for a font.

I.3 View Morphing

The View Morphing technique was introduced by Steven Seitz and Charles Dyer at SIGGRAPH 1996[2], providing a transition between two images in different pose or viewpoint. This project makes use of this technique in two ways described in sections I.3.1 and I.3.2.

I.3.1 View Morphing in Fingerspelling

Visualization of fingerspelling could be enhanced with view angle rotation so that the part of the hand that is otherwise partially occluded could be fully realized. Examples of letters where view rotation is most useful would be the letters "p" or "q", Figure I.1.

Another purpose to View Morphing is to smooth the transition between images that were taken from different recordings, but are being "spliced" together to look like a continuous stream. To this end, visually apparent discontinuities caused by imperfections in the data are minimized.

I.3.2 View Morphing in 3 Steps

The Figure I.5 can be used to describe the 3-steps in view morphing. Images I0 and I1 represent "snapshots" of two images of the same target but from different camera views.

The first step, the prewarp step, uses a matrix H_0, a 3x3 matrix that specifies the position and orientation of the $Image_0$ plane in world coordinates. The Image I'_0 is is created by applying H_0 _inverse to Image I_0 and likewise for Image I'_1. The second step is the image morphing step, which can be performed by any standard morphing technique, producing Image I'_s. The third step is the postwarp step the maps I'_s back to the desired position and orientation by applying H_s, creating image I_s.

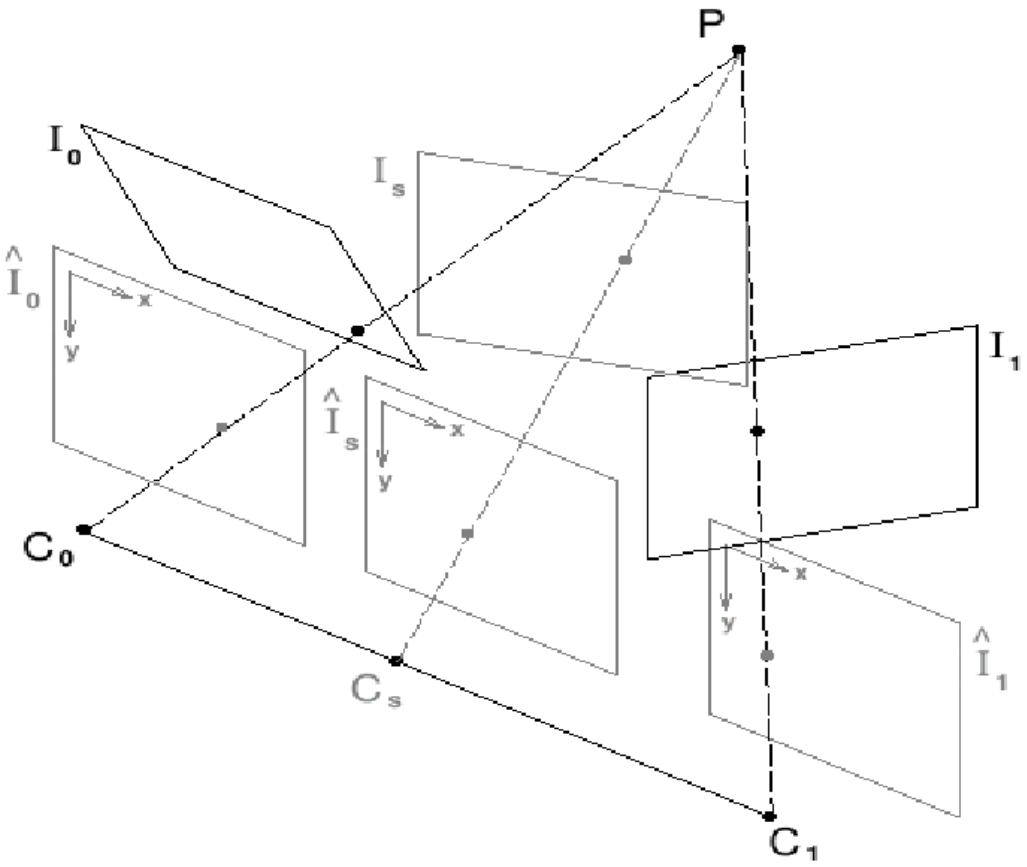


Figure I.5 3 Steps of View Morphing

I.3.3 Fingerspell Morphing Visualization Examples

The following 3 sections present images generated by the View Morphing technique.

I.3.3.1 Rotate View Angle for a Virtual Camera

The following two figures, Figure I.6a, and I.6b, are sequences of images produced from View Morphing software using the 3-step technique. The first

sequence of images, Figure I.6a, starts with hand images from a two-camera capture, taken at the same moment, but from different camera angles. The first arm image, and last arm image, are the only ones that were from a camera. The images in between, were produced by the View Morphing software from openCV[4].

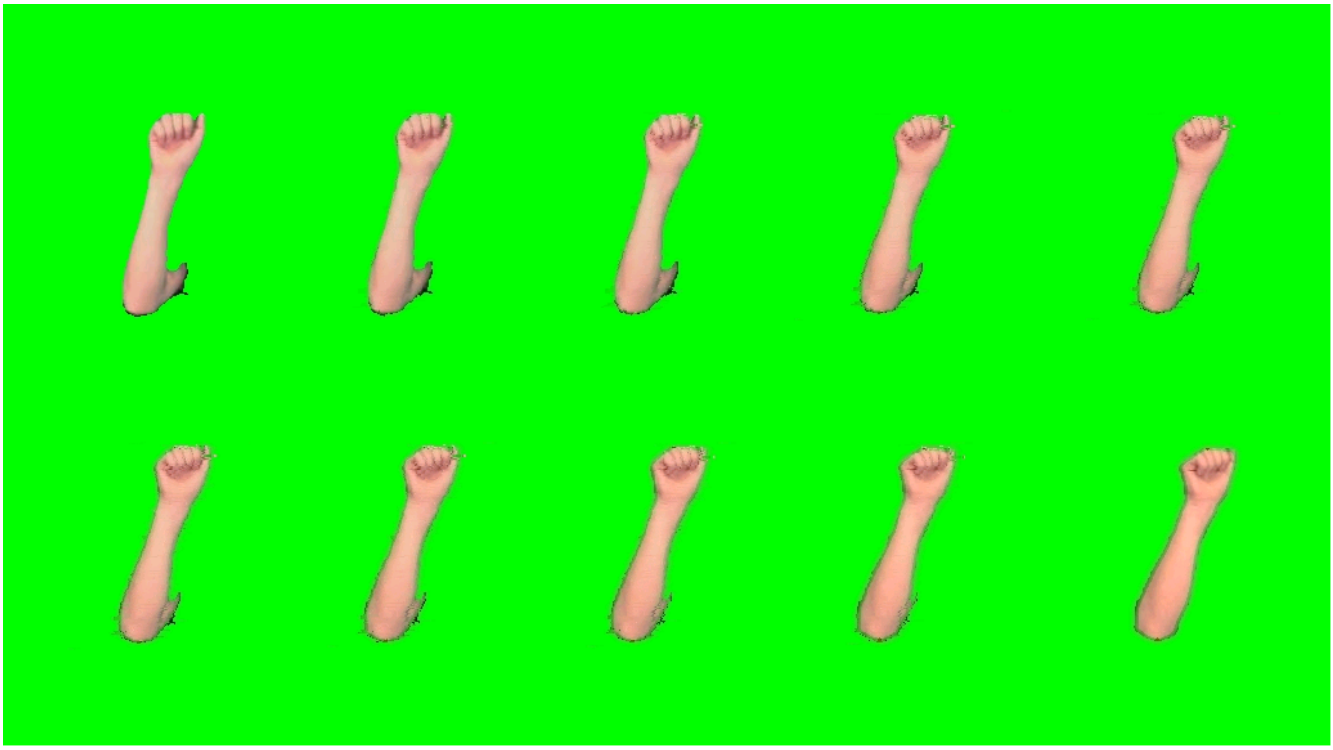


Figure I.6a Morph between two views

I.3.3.2 Smooth Transitions Between Bases

The next sequence, Figure I.6b, shows smoothing transitions between separate recordings through the use of View Morphing. The two images at the front and end are the input to View Morphing. All of the images in between were produced by the View Morphing process. (The difference between the images is the arm rotates and the hand turns, gradually showing more fingers.)

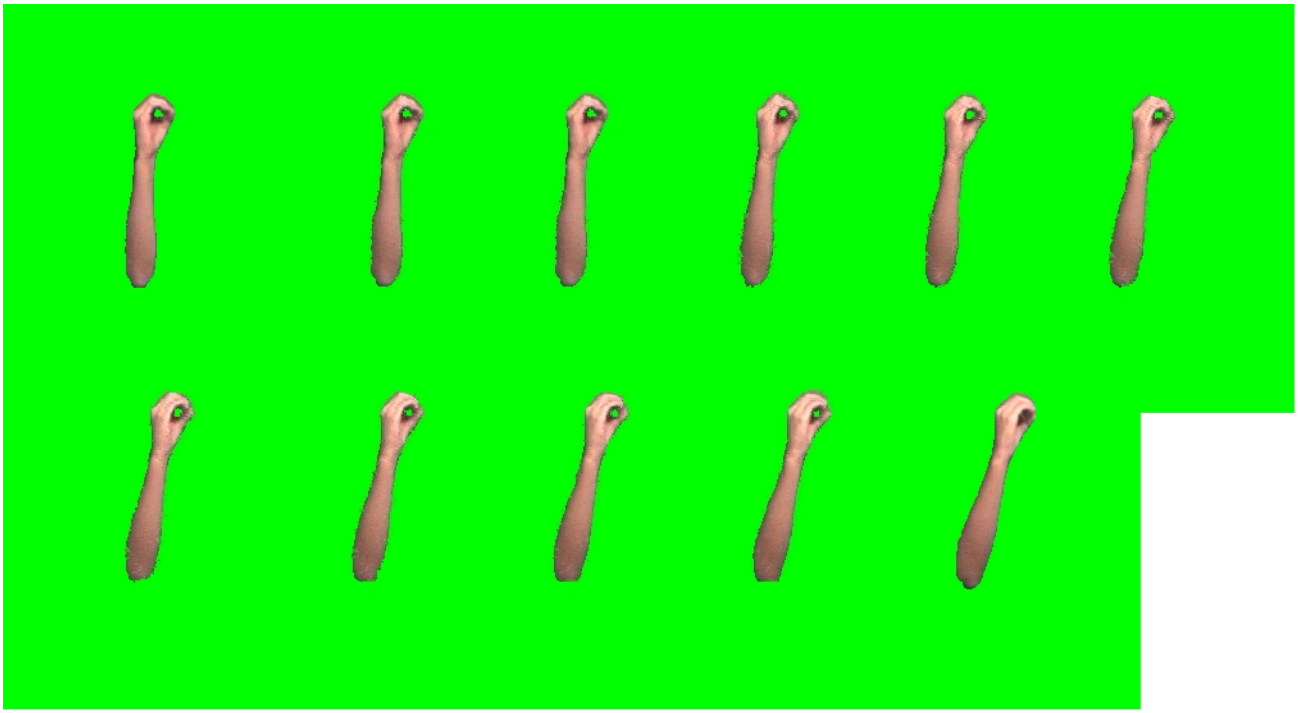


Figure I.6b Morph between different Recordings

I.4 Continuing Research

There are at least three areas of continuing research. Each of the following sections describes a possibility.

I.4.1 Pure Computer Science & Engineering Research

A possible extension to this masters project would be using the 2-D images as input to pattern recognition software and associating the images with hand positions on a 3-D model of a hand. The database would then be replaced by the movement patterns required to form each letter relative to the 3-D hand model. The benefits of this database is that the hand would then be scalable to any size and viewable from any angle. There is comparable research in regards to ASL, (see http://www.bu.edu/asllrp/).

I.4.2 Fingerspelling Art

Visualizing the 3-D model version of the fingerspell database would be interesting from a purely artistic standpoint. The database would more then likely end up as just hand bones. If so, then "skin" could be projected onto the bones. Not having skin would allow a user to view through the "skin" so more hand positions would show through at various angles, normally occluded. The fingerspelling of such a 3-D model, while rotating around the z-axis, would be fascinating to watch. Skin, scale, and tone changes would provide added expression to the words that are being spelled out.

I.4.3 Educational and Psychology Research

Such a database could be used in an educational or psychological study, collaborating with a computer science department. Possibilities for this area extend far beyond the examples given in Section I.2 into studies and styles of learning for exceptional children and leaning through play.

II. Approach

Finding the "best practice" was an iterative process. This section illustrates the final processes used from data capture to visualization.

II.1 Contents of Image Files

All of the software applications work internally with a string of bytes when manipulating the data. The software considers the extension when attempting to read in or write to files. The difference between handling different file formats only occurs at the I/O interfaces. Figure II.1 illustrates this relationship.

II.1.1 Working Units

The database is not saved as jpeg files because the jpeg compression algorithm loses data. An alternate, non-compressed, binary storage is used. This storage is not good for general use. It is acceptable in this application because there is not a lot of hand data per image. Most of the pixels are not part of the hand. The name pnk is used as the file extension name because in the task to remove non-hand data, if was found that only pixels more red than green need to be stored. The data lost by jpeg compression is on the border of green and red. The borders tended to be interpolated at each jpeg save, prior to the use of pnk files.

This, "pnk" format is the following:

```
string version number and identity
int width
int height
short x_value[0]
short y_value[0]
char r
char g
char b
char alpha // to keep nice boundaries
short x_value[1]
short y_value[1]
...
```

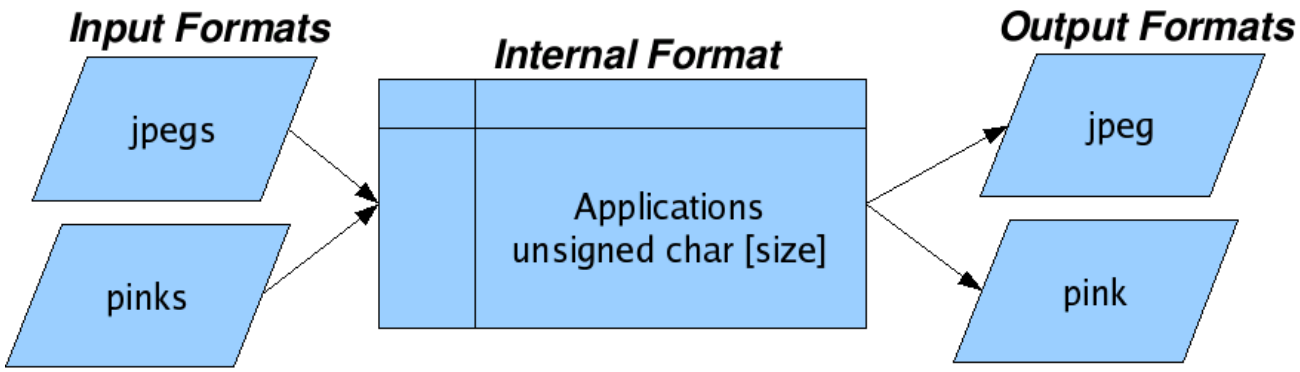


Figure II.0 Working Units: jpegs and pnks

II.1.2 Base Image for Hand at Rest

The following figure, Figure II.0, is visualization for the hand in the rest position.

Figure II.1 - Hand in rest position, "base.pnk"

II.1.3 Base Images for Each Letter

The following figure, Figure II.2 is the fingerspelling for a,b,c, and z.

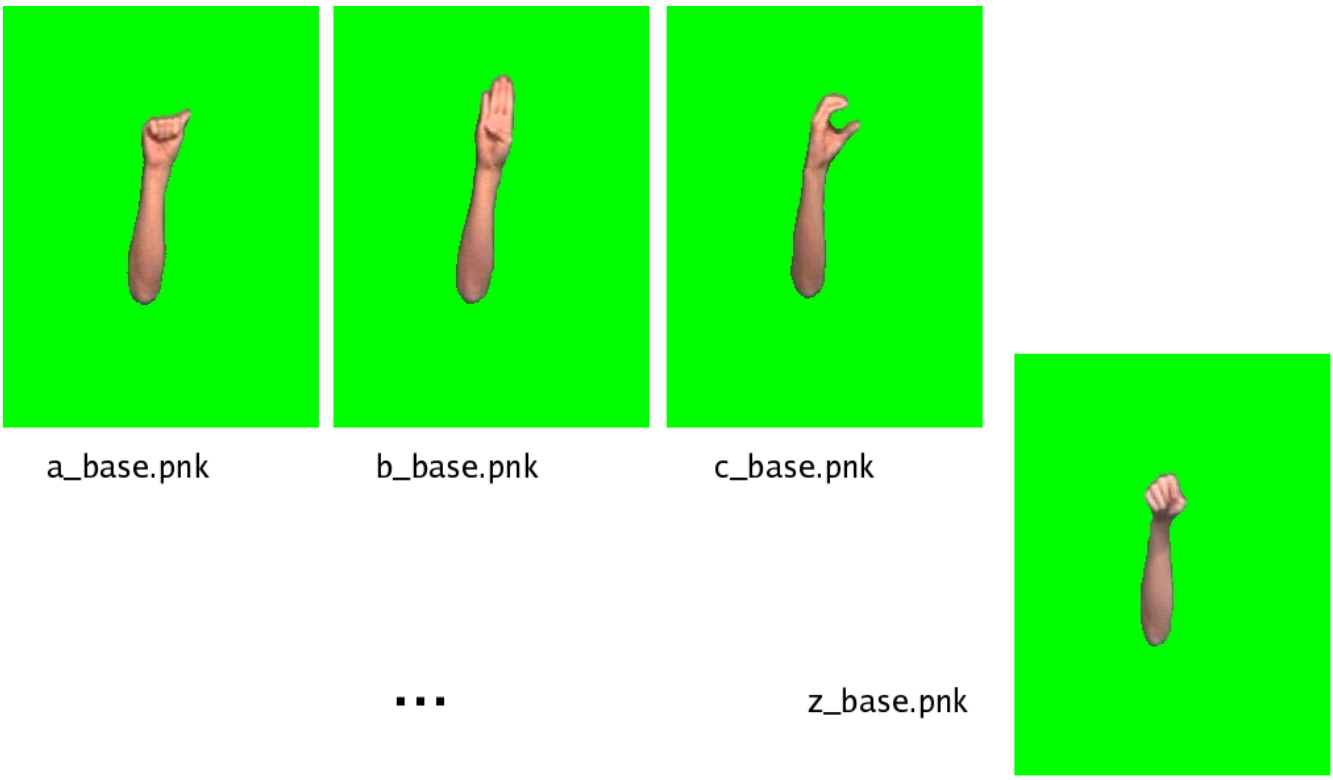


Figure II.2 - Fingerspell bases for a,b,c, and z

These files are the starting file in directories of 2 letters sequences, such as letter "g", in directory "ge" when the letter is first in the 2 letter combination. Likewise it is the ending file in directories of 2 letters sequences, such as letter "e", in directory "ge", when the letter is last in the 2 letter combination.

II.1.4 Singe Letter Transition Example

The following figure, Figure II.3 is a partial visualization of forming the letter "b", by itself, from the rest position to the letter "b" sign, back to the rest position.

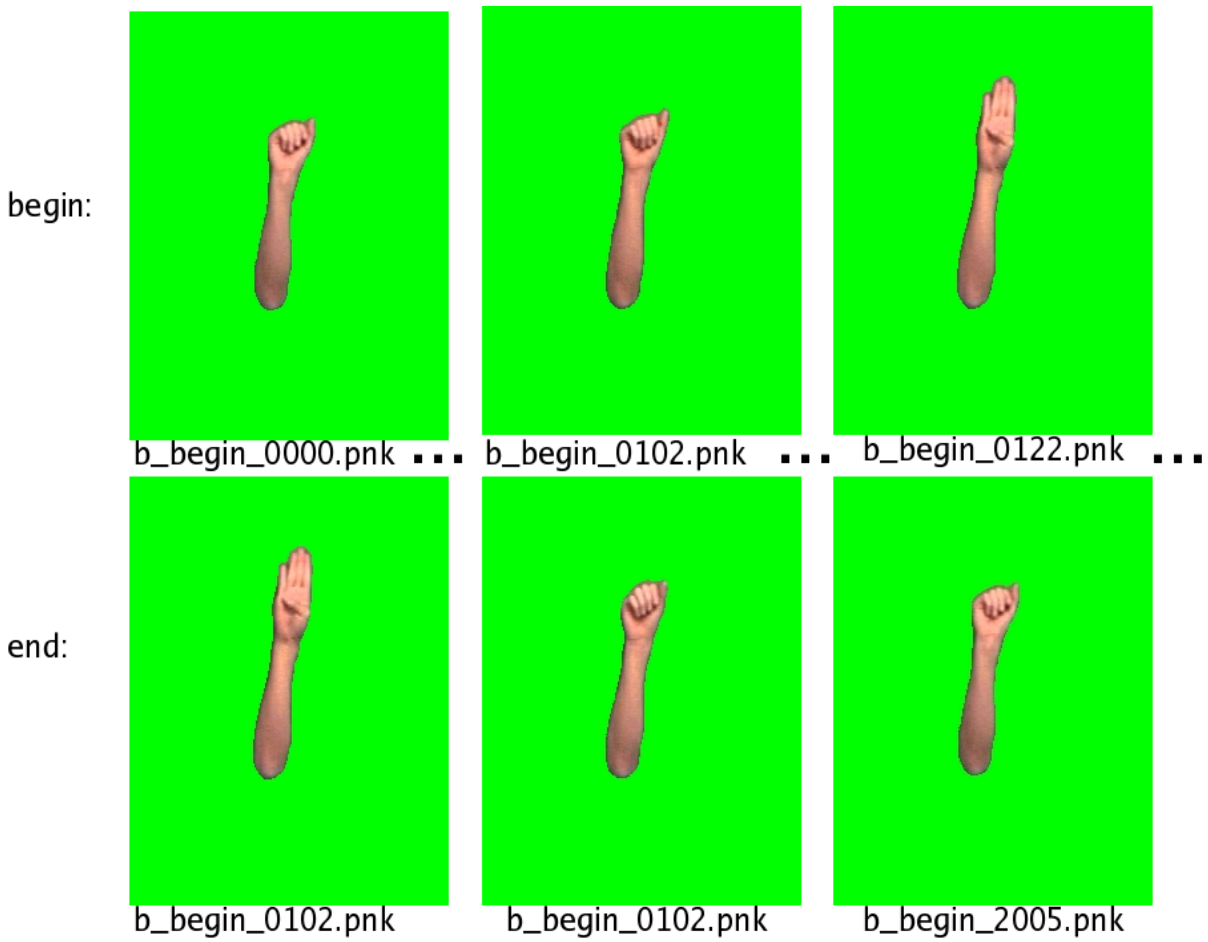


Figure II.3 - forming the letter "b"

II.1.5 Two Letter Transition Example

The following figure, Figure II.4, starts at base "g", morphs to the captured version of "g", proceeds to "e", (all the files in between is represented by the symbol "..."), and then a morph is done from the captured "e", to base "e".

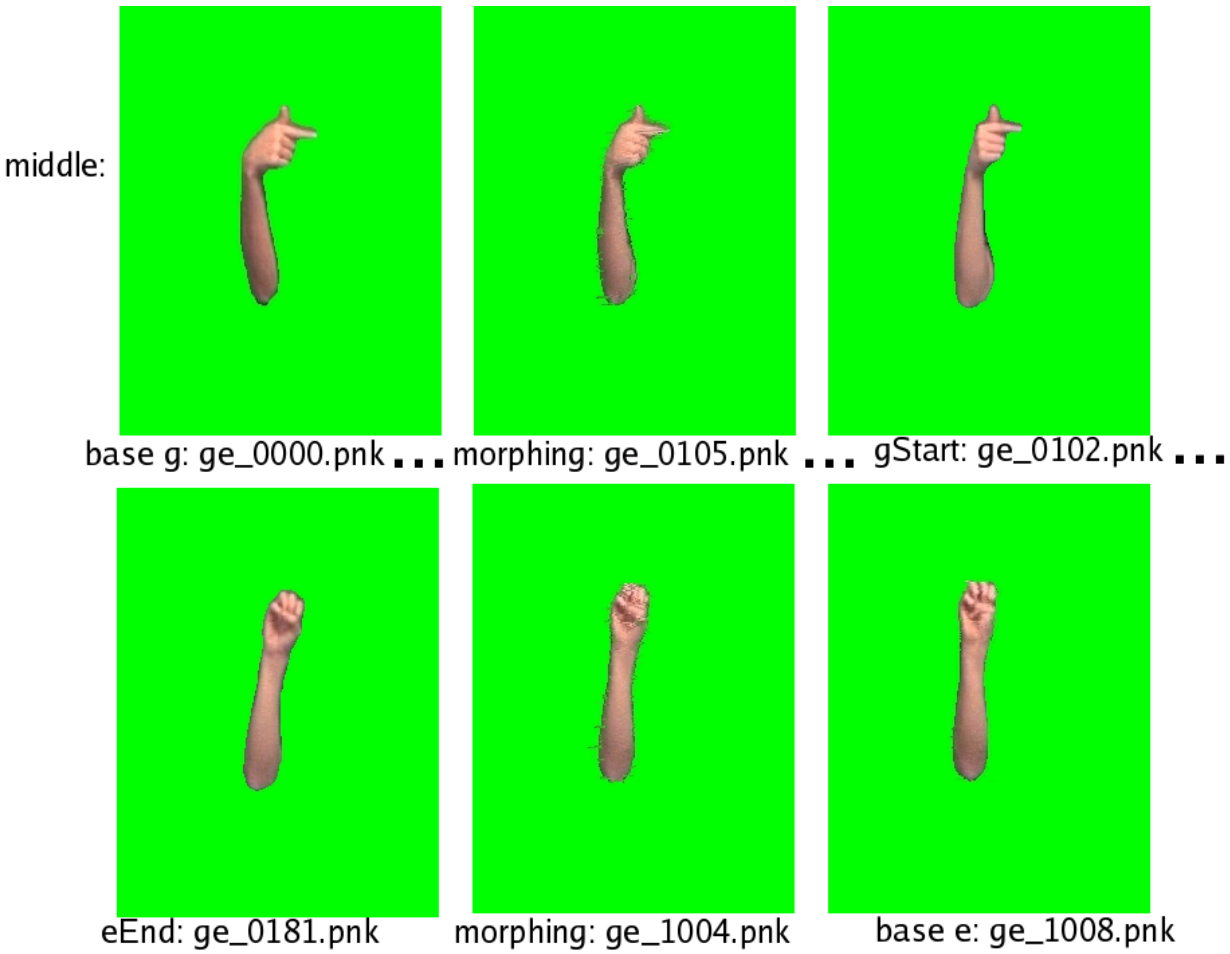


Figure II.4 - The files forming sequence "ge"

Notice the numeric value used as part of each name. The file with the number 102 is the starting file for the sequence from a different recording, that was not morphed.

II.1.6 Word Transition Example

The word book is formed from a concatenation of all files in certain directories. Figure II.5, is the directories needed for the word "book".

b_only/begin bo/middle oo/middle ok/middle k_only/end

Figure II.5 - File order for the word "book"

The section II.3 details the contents of each of the directories.

II.2 Video Camera Data Capture

Video data was collected with the signer standing behind a green curtain with only the arm exposed. This data was a 1-camera capture to video tape. The entire alphabet was captured with all combinations of two letters from one camera.

The data tapes were processed on a linux box, running red hat 9.0, using the open source program, "kino"[5], from sourceforge.net. This program creates jpegs from video. The program allows a user to take excerpts from the tape, so the data was minimized to only the amount that appeared useful.

II.3 Fingerspell Database

Other than running a program that converts pnk file to jpeg files, the images shown in Figures II.1-II.4 are taken straight from the database. Section II.3.1 outlines the requirements that the database needed to be met in order to be usable by the application software developed for the project. Sections II.3.2-II.3.4 describe the directory structures. The last section, II.3.5 is pseudo code to initialize the database when using the application software, fsvis.

II.3.1 Fingerspell Database Requirements

The Fingerspell Database has been purposely put in a format that allows the developer to know what data a file contains by the name of the file. Furthermore, the file may be individually inspected and modified. Part of each filename is a number that indicates the file order in the sequence of files in the directory in which it is located. The data image represented by each file is of one hand only, in one particular Fingerspell position, or in motion to, or from, a Fingerspell position. The number of files comprising any sequence of fingerspelling must be sufficient to get realistic playback.

II.3.2 Database Director Structure - Top Level

The following figure, Figure II.6, is the directory structure at the top of the database.

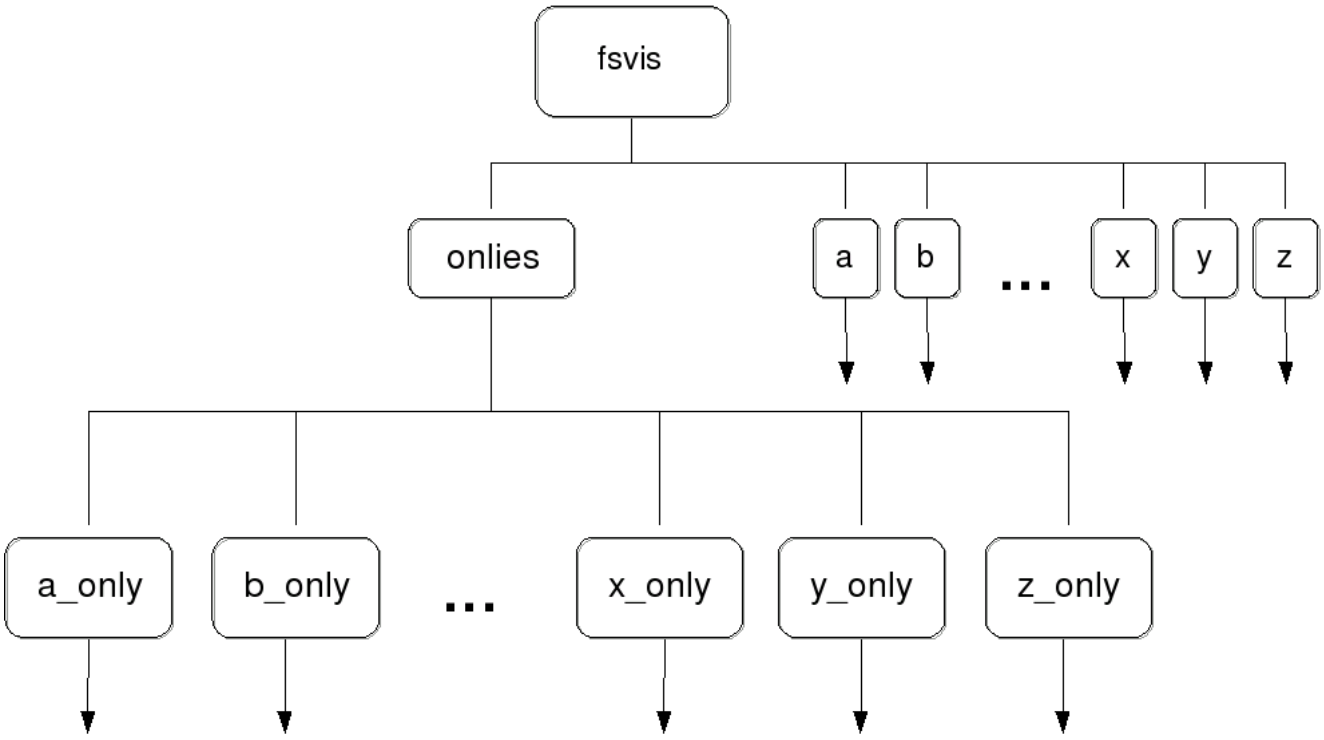


Figure II.6- Top of the Database

II.3.3 Database Single Letter Directory Structure

The following figure, Figure II.7, is an example of what the directory structure looks like under the "onlies" directory. It is the "b_only" directory, under "onlies", under "fsvis".

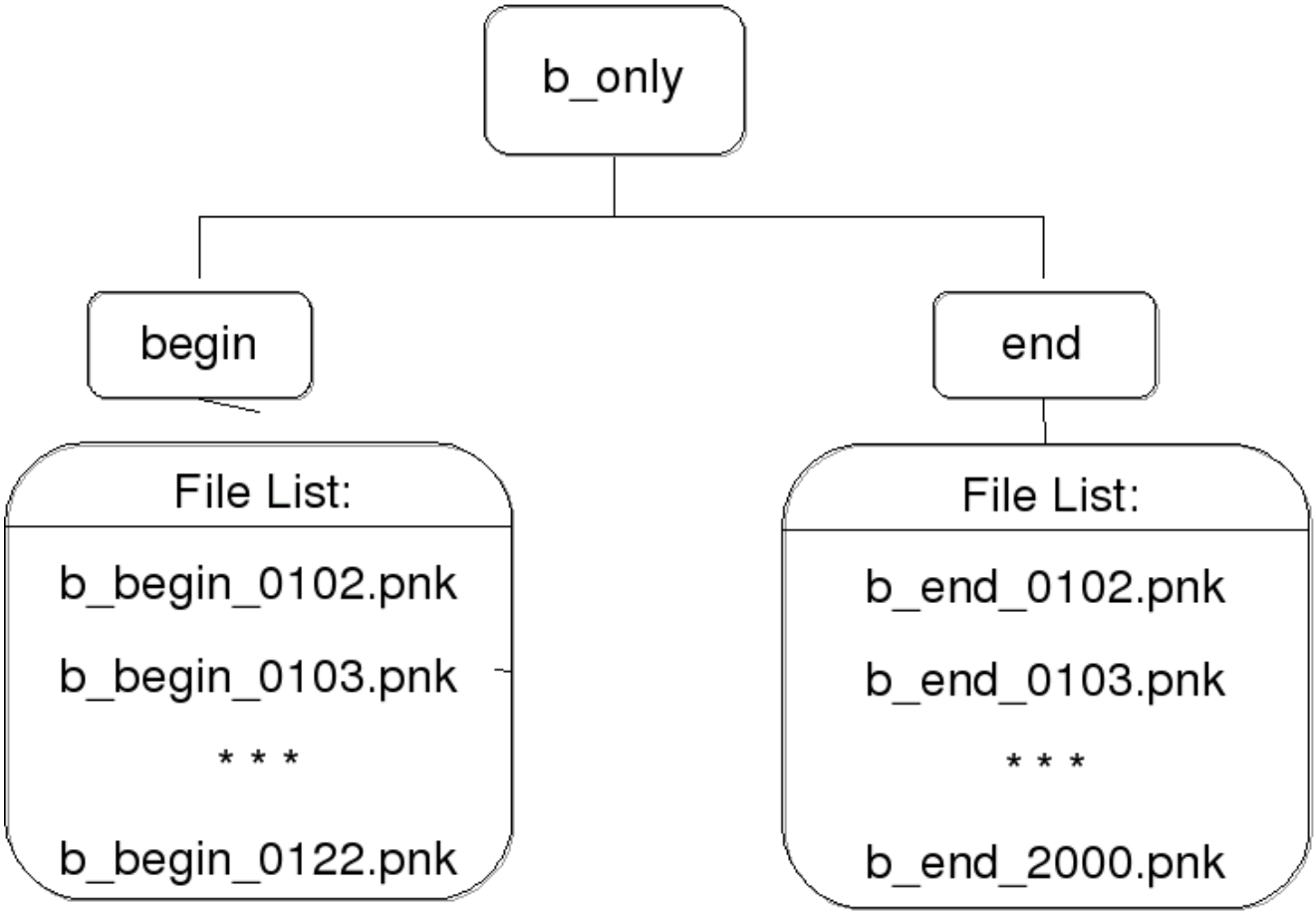


Figure II.7 - Example of the b_only directory

Consider the forming of the letter "b" in fingerspell, (see Figure I.1). The hand is at an "at rest" position, and moves to form the letter "b", then goes back to the at rest position. The files in the the directory "begin" visualize the sequence from the rest position to the forming of the letter "b". The files in the "end" directory go from the hand forming "b", to the rest position.

II.3.4 Database Two Letter Directory Structure

Going back up to the directory "fsvis", there is one directory for every letter in the alphabet. The following figure, Figure II.8 is an example of the directory structure for the letter "a".

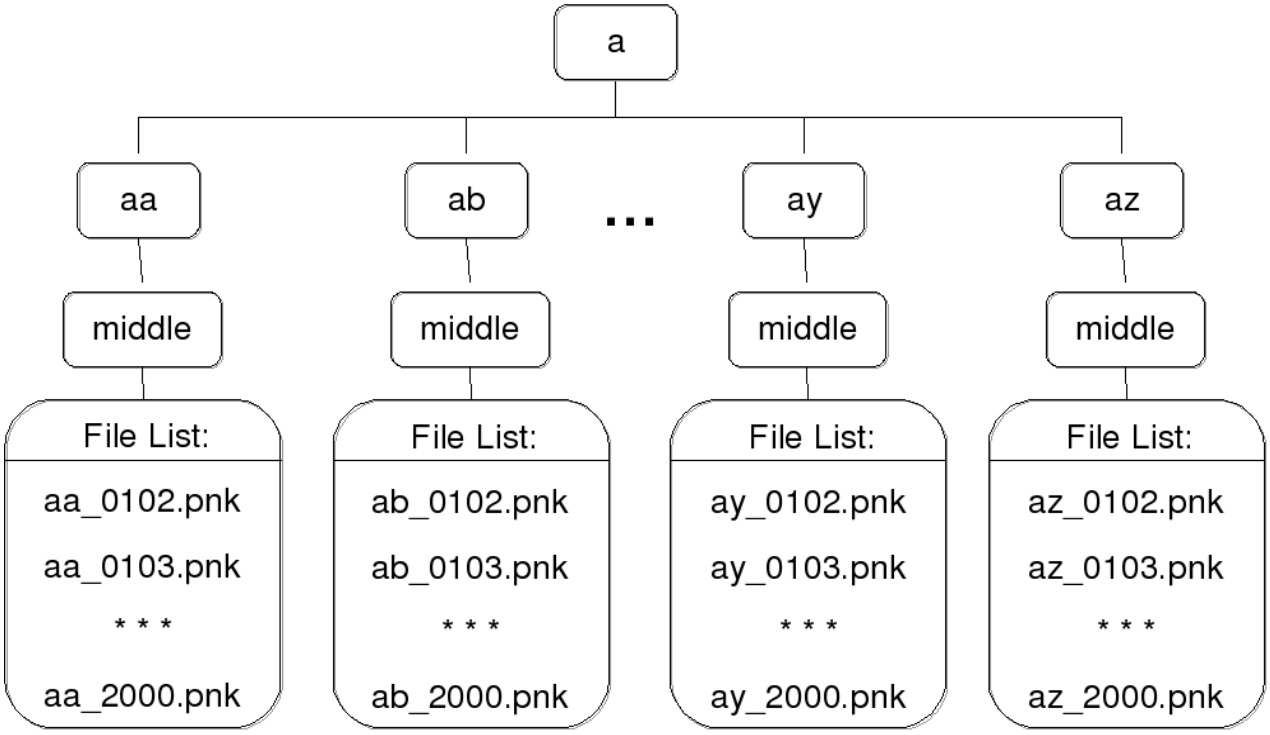


Figure II.8 - The "fsvis" database directory structure under "a"

Now consider the two letters, "or", in the middle of the word more. The fingerspelling for the "or" transgresses directly from the "o" to the "r" without going to the rest position. The rest position, at the beginning, starts the sequence of files for the letter m, and ends the sequence of files for the letter r.

Refer back to Figure II.5, and notice the order of the names of the directories needed to fingerspell the word book. When the first or last letter is being formed, one of the "?_only" directories are used. For the first letter of the word, the _only/begin is used. For the last letter of the word, the _only/end is used. All of the letters in between are taken from letter combinations. The letter combinations only have a "middle" directory from which every file is used.

II.3.5 Database Initialization

If the user runs applications while inside the fsvis directory, all of the files in all of the folders are either read-in or are eligible for processing. The

following figure, Figure II.9, is the recursive pseudo code that is frequently used for initializing the data base.

```
ProcessFilename ( Filename )

if Filename is a Folder
then
    Loop for in_filename in Folder
        ProcessFilename ( in_filename )
    end_loop
else
    PictureInfoDB.append(new PictureInfo(Filename))
endif
```

Figure II.9 Pseudo code for initializing the Database

III. OpenCV and Morphing Theory

III.1 Fingerspell Morphing Requirements

In order to achieve a two angle view of a single hand position in fingerspelling position, data had to be captured from two-camera at slightly different angles. The view morphing was done using routines from the openCV[4] library. A means of displaying both the input images and the virtual, morphed image, was developed. A slider was provided to vary the angle between the two input images along with a means to manipulate and

display the corresponding points. From these points the Fundamental matrix was calculated.

III.2 View Morphing Theory and Characteristics

The View Morphing Theory was created by S.M. Seitz and C.R.Dyer. It was documented in a SIGGRAPH, '96[2] paper of the same name. The algorithm involves 3 steps: prewarp, image morphing, and postwarp, as seen in the previously presented figure, I.5. and described in Section I.3.2. One of the interesting characteristics mentioned in the SIGGRAPH paper is that the Pre-warp images each have a set of corresponding points on the same scanline, so one scanline at a time may be calculated. Frequently the same Fundamental matrix can be used for different images. But, when objects appear in one image and not the other, they will appear as ghosted in the morph image. Another fact is that View Morphing does not work so good when the depth of the image is very large compared to the depth of the object.

III.3 OpenCV and View Morphing

> "The Open Computer Vision Library is a collection of algorithms and sample code for various computer vision problems. The library is compatible with IPL and utilizes Intel Integrated Performance Primitives for better performance." [4]

Software that uses the openCV, morphing procedures, was developed with a 3 image display, Figure III.1. On the left and right displays are the images used as inputs to the morphing procedures. They are pictures of the same object viewed from two different angles. The middle display is the virtual morphed display. A number between 0 and 1 represents the virtual angle between the left and right image. A slider is available to interactively view the morphed image as it changes with the corresponding change in the virtual angle.

III.4 View Morphing Support Development - morphPoints

The Morphing technique requires two sets of corresponding points in which an index, i, for points1[i] is the target pixel, in the first image, and points2[i] is the x,y-position of the same target pixel in the second image. These point arrays are used by the openCV Morphing software to calculate the fundamental matrix. The set of points is not the same for all the images. The morphPointsG allows the user to add points that associate a pixel on the left image with a pixel on the right image. Sets of points can be read in, displayed, created, modified, and saved. The change in the set of points is followed by a calculation of the fundamental matrix and the corresponding difference in the morph.

The user selects two corresponding points through a browser. When a pair is selected a red dot appears on the left and right image at the corresponding point position. The user can the move the points using the arrow buttons. Figure III.1 shows the morphPoint application, with some red dots at the corresponding point locations and the morphed image as the middle display.

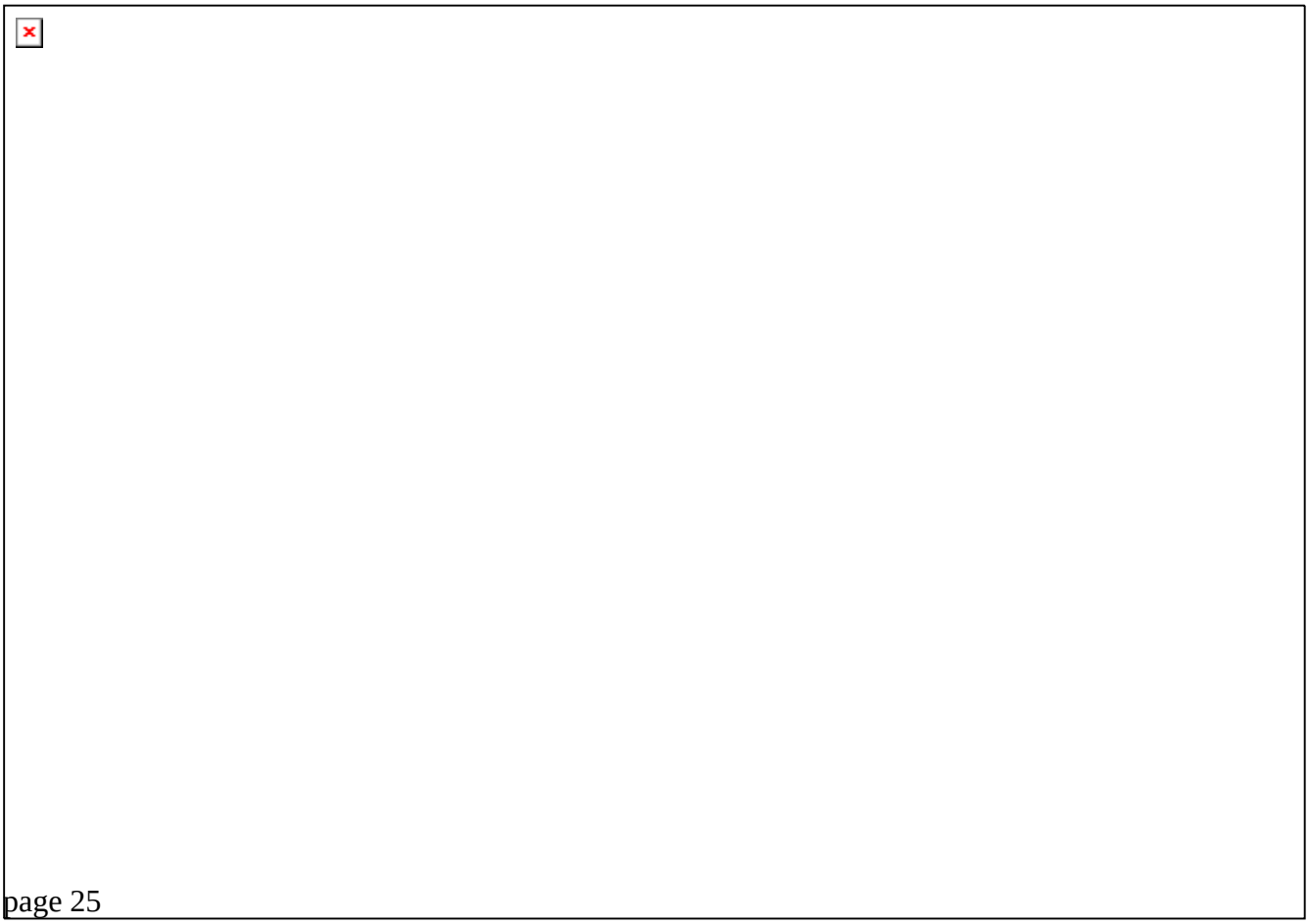


Figure III.1 morphPoint Application

IV. Application Software Architectural Design

This section documents the requirements and architectural design of the applications used to create and display the database.

IV.1 Image Data Modification Software Requirements

The data was captured using a Video Camera. The original jpegs, created by a linux software program, had dimensions: 720 x 480. These files were changed to dimensions: 240 x 320 pixels. Many more image modifications were necessary. Normally, the changes needed to be applied to a set of files. The decision was made, early in the project, to develop software that has an automatic feature to apply changes to all files in all sub-directories. Although there are many applications that perform the same type of tasks required by this project, the components of this software are tailored for this project. In particular, a modification made to a virtual image of the data from one particular file can be applied to all files in all sub-directories. Such modifications are: resizing without rescaling, eliminating unwanted data, translating, scaling, brush strokes, morphing, etc.

Another application, bfit, was developed that runs similar tasks, but in a batch mode. This application performs a set of tasks to requested files. The image manipulation tasks are listed by command name, each command contained in the same file, supplied as an input parameter to bfit.

IV.2 Rapid Prototype Development

The software development requirements for this project, across applications, had many overlapping components. Instead of duplicating the software and making maintenance more complicated, a library of common modules was created. Since most of the components are related to User Interface, the library was named the Nice User Interface Toolkit or Nuit. Once this library was constructed, the application drivers were simplified. Any new

development of applications from that point on was done in a significantly shorter amount of time. The following sections describe this library and the application requirements to use the library.

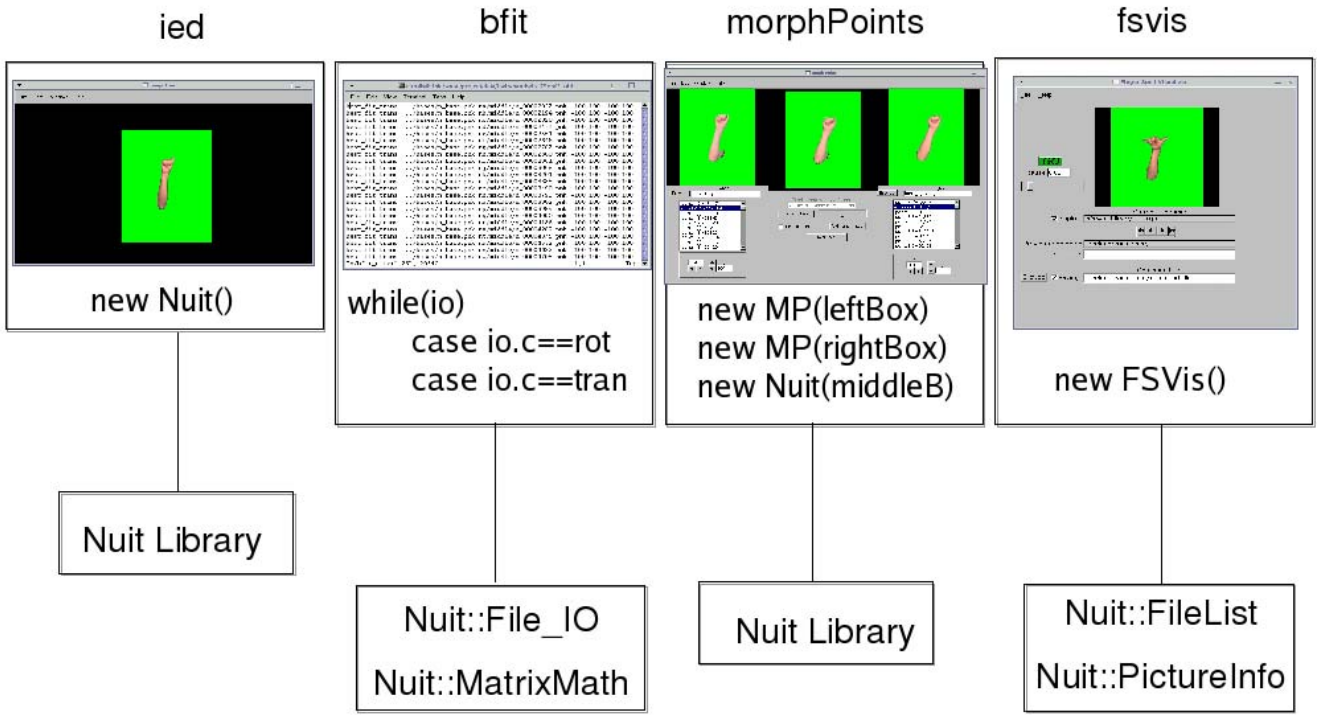
IV.2.1 Nice User Interface Toolkit (Nuit) Library

The main use for the nuit library is to provide edit functions for the virtual window display. Each application creates its own main window and its own menubar. The menu buttons may be added to the application's menubar with a reference to the corresponding Nuit class procedure. The application can then only use the classes for which there is interest. Alternatively, the application can use the nuit library as a utility library only.

IV.2.2 Simple Application Drivers

The following figure, Figure IV.1, is the control logic for the top level of the application drivers. Figure IV.1 shows that the program that edits the images using a virtual window, ied, uses most of the Nuit Libary, where as the bfit program has no windows and only uses the File_IO and matrix routines from the library.

Figure IV.1 - Application Drivers



VI.2.2.1 iedG

The need to modify the image data caused the development of the iedG application. This application uses all of the components of the nuit library. The software was used to modify all of the images starting from the original jpegs to the current state of the fingerspell database.

VI.2.2.2 bfitG

The exception to iedG having all the code used to create the database, is that some procedures were broken out so they could be run in a batch mode.

VI.2.2.3 fsvisG

The fsvisG program expects the database file structure described in section III. The program provides an input field for the user to type in the User Request String, (URS). The string is then visualized in the virtual window. Another input field is provided so the user can name a file that will contain all of the files used to visualize the URS. If a subdirectory is created using the same name as the name of the "command file", list of filenames, the script run_pnk2jpeg, (and pnk2jpeg.cpp) will read the pnk files, convert them to jpeg files in the subdirectory, naming them in sequential order so the mjpeg foundation software can create a mpeg file. If the script file, do_movie is run in the directory containing the jpegs, and given, as input, the name of the mpeg, it will create the movie. The contents of the do_movie file is:

```
jpeg2yuv -f 25 -I p -j fsvis_%04d.jpeg | mpeg2enc -o ${1}.m2v
```

This assumes the mjpegtools software has been installed. Once the software is installed the man pages will give details on the parameters used in the script file, pnk2jpeg/do_movie.

IV.2.3 Event Handlers in Nuit Class

The nuit component of the software attempts to allow quick access to sequential files by using an idle proc that is checking that at least the next 3 images have valid data pointers. If any of the next 3 images have invalid data, then the data is reread. The idle proc minimizes the use of memory without losing the speed advantages. There are many more events, but these

are mentioned because all sub-folders are searched for either jpeg files or pnk files. Keeping all the data in memory at once would not work. Figure IV.2 is the flow chart of the idle proc and user interface buttons .

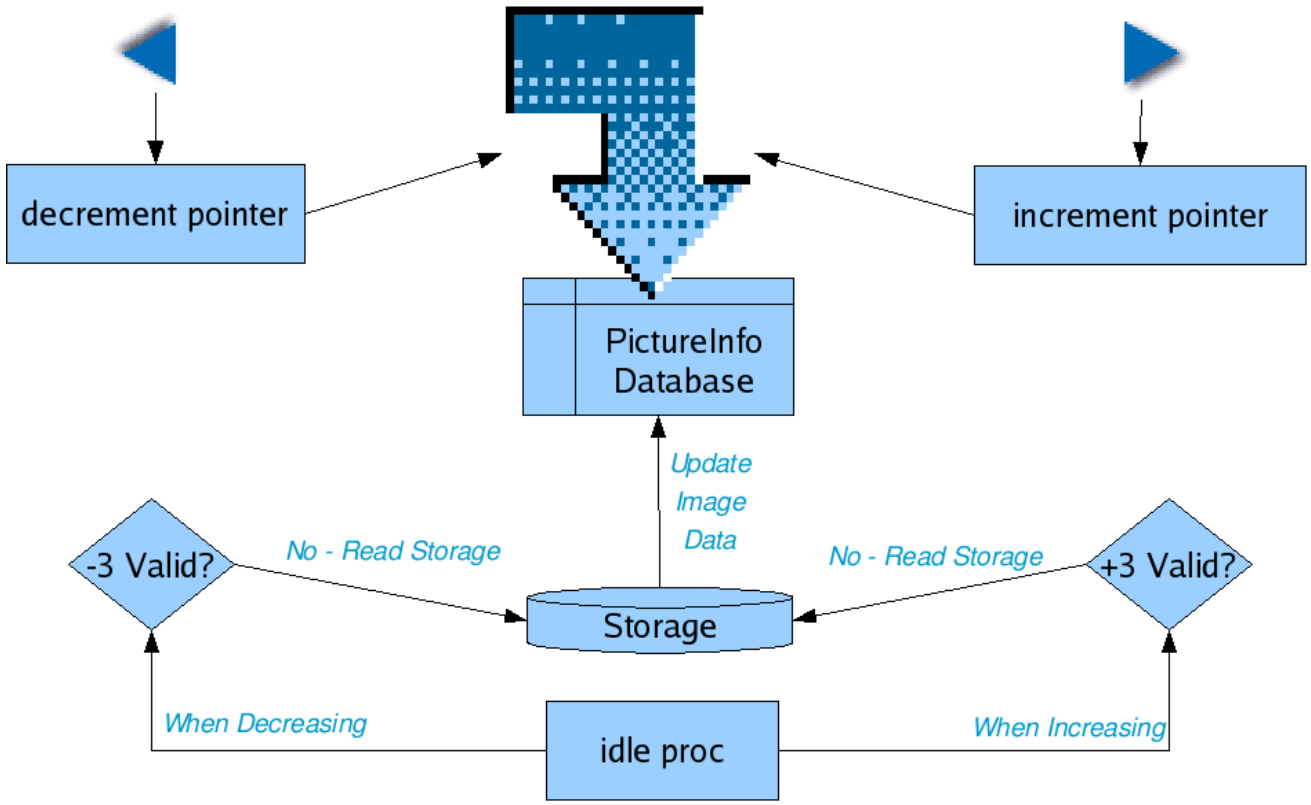


Figure IV.2 - Event Handlers in Nuit Class

IV.2.4 Virtual Window Nuit Architectural Diagram

Most of the application programs make use of a menubar and a virtual window to display the state of the current image. The Figure IV.3 is the Architectural Diagram for these types of applications.

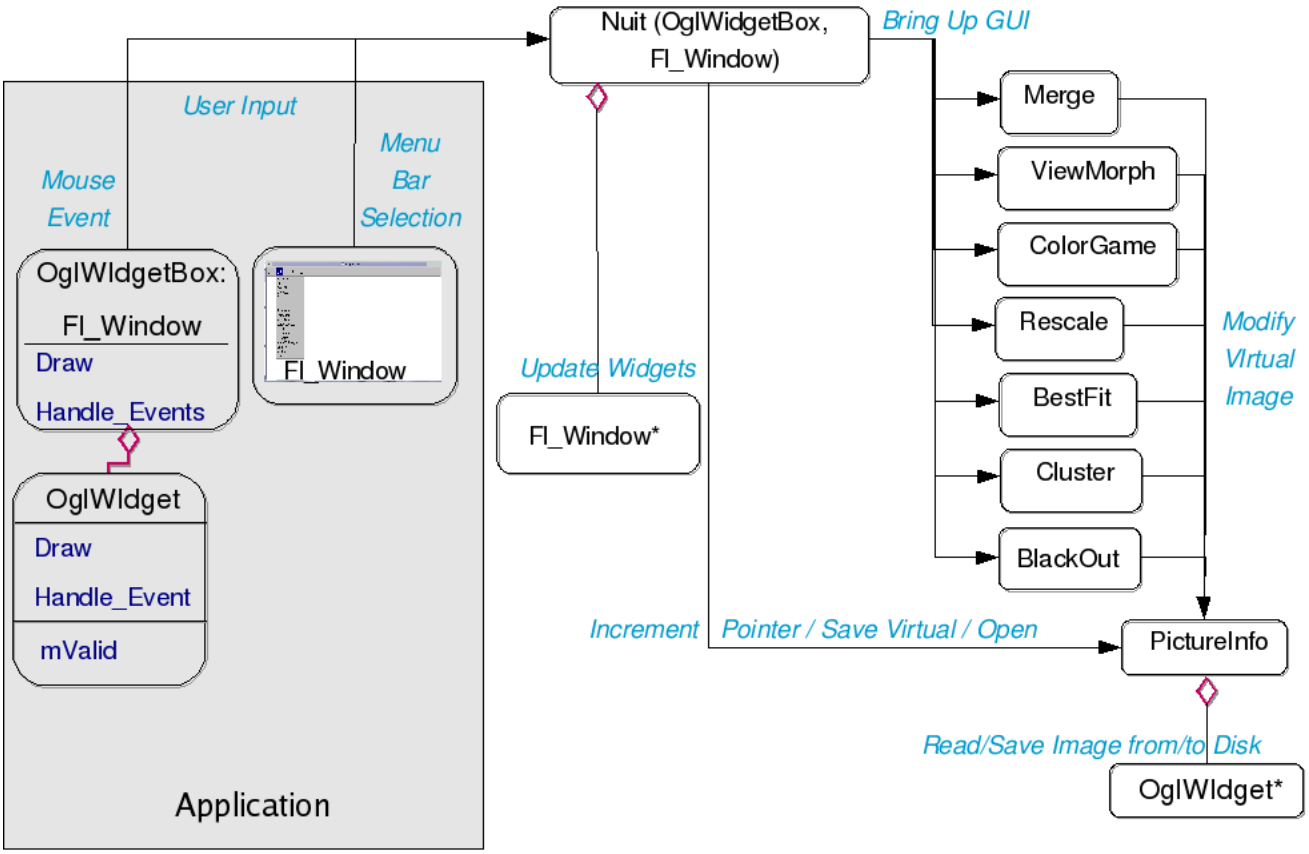


Figure IV.3 - Architectural Diagram when using Virtual Window

IV.2.5 Batch Nuit Architectural Diagram

The bfit program has no windows and only uses the File_IO and matrix, so the architectural structure is simpler then the structure for the virtual window. Figure IV.4 shows the Architectural diagram for this situation.

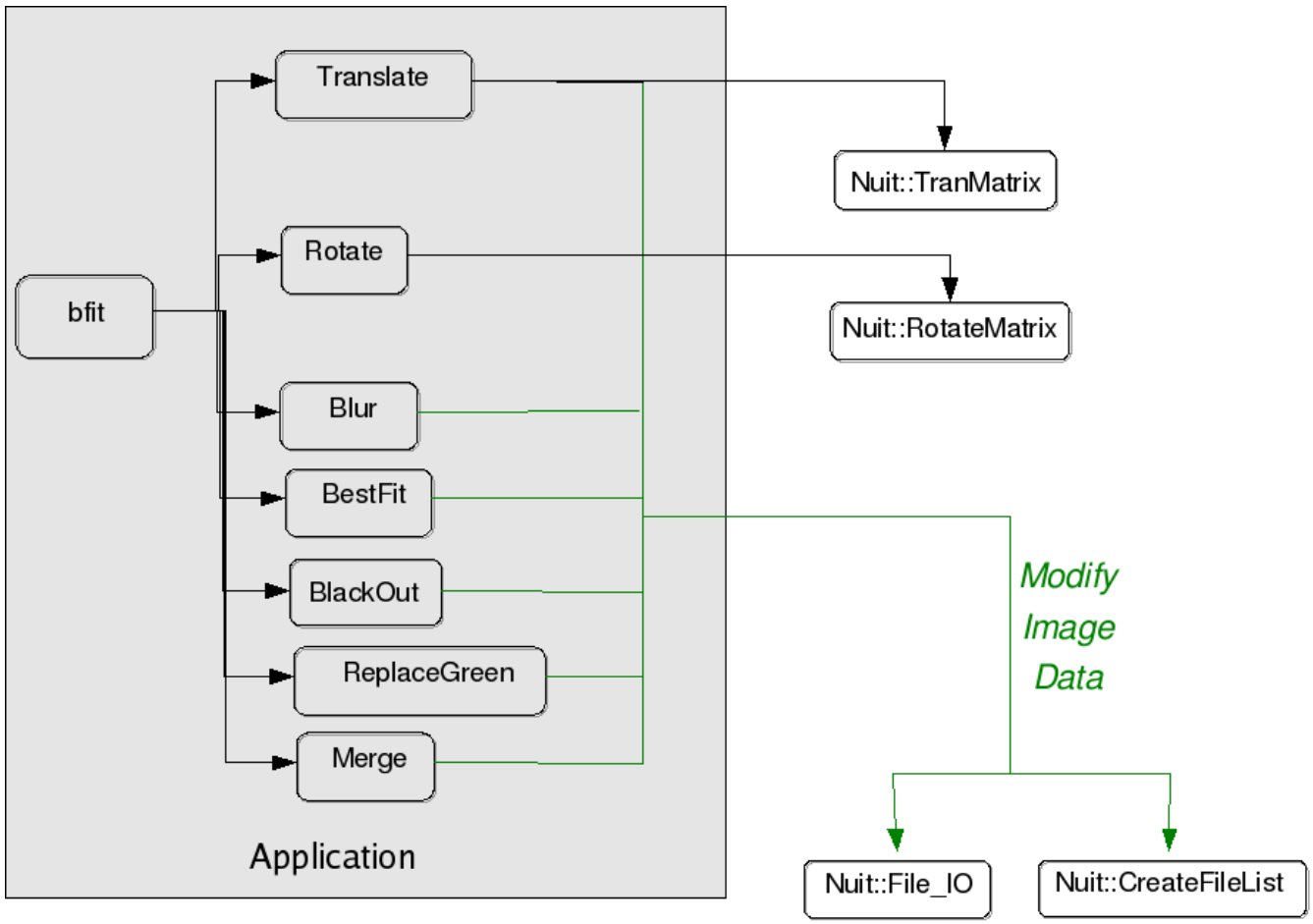


Figure IV.4 - Architectural Diagram when Batch Mode

IV.3 Software Requirements Specification

1.0 Language

All software must be written in an object oriented language such as C++.

2.0 Data Base Format

An external database must be provided with image data of hand sequences that can be combined in an efficient manner to visually represent the User Request String, (URS). These requirements have no mapping onto the Architectural Diagrams because they specify what the data will look like instead of the design of the software.

2.1 The database must be in the form where each file is named by the sequence of letters it visualize.
2.2 The alphanumeric order of the names of the files must be the same as the order displayed
2.3 Each file must contain the image data that only represents the hand. All other data must be removed.
2.4 Each letter combination will have a sufficient number of files to allow a realistic playback.

3.0 Modification of Image data - The ied, bfit Applications

A Editing applications must be developed to modify the original data images to achieve all the database requirements

3.1 The application must be able to read and write both jpegs and pnk files. (File_IO)
3.2 The application must be developed that provides a virtual display of the requested changes made to the original image. (OglWidget)
3.3 All components developed to modify a virtual image should allow a user to make the corresponding changes to groups of files. (All Button Provided on appropriate GUIs)
3.4 The user must have complete control over which files will be changed. (All files in all sub-directories are changed when an "All" button is hit.)
3.5 The application must provide the following image modification functions:

translate, rotate, blur, color modification, find x,y deltas to best fit another image, merging with a background, brightness, rescale, change dimensions, and scale to fit another image. (Each function is mapped to a class)
3.6 Editing procedures must be provided that use the morphing software to improve the database. (ViewMorph)
3.7 Editing procedures must be provided that modify the morphed images to eliminate artifacts of the process, (Cluster).
3.8 An Edit application must be developed that allows batch processing of image modification (bfit)
3.8.1 The input file to the batch mode version of the edit program must specify the image filenames that are to be modified and all specified module's parameters.
3.8.2 The batch mode version must be able to read and write both jpegs and pnk files. (FileIO)

4.0 Morphing Wrapper - The morphPoints Application
An application must be developed to manipulate the parameters used by the openCV morphing procedures. The two images morphed may be either for the purpose of morphing two viewing angles, or to demonstrate the smoothness of the transition between a base letter and the first letter of a two letter combination.

4.1 There must be a capture of a hand forming a particular letter in Fingerspell, (FS), from two different viewing angles (Database Requirement)
4.2 The two views may be morphed using existing software, such as the openCV library.
4.3 The morphed virtual morphed image must be displayed. (Nuit::OglWidget)
4.4 Both Left and Right displays of the two images used as input to the morph procedures must be displayed. (FSVis::OglWidget)
4.5 A widget must be provided allowing the user to change the morph viewing angle to an angle in between the Left and Right Images' viewing angles. (Slider)
4.6 The two data point arrays that associate target pixels by index, must have

the values be displayed on the Left and Right images in the target location. (Red Dots)
4.7 The widgets must be provided that read in (Open), display (Browser), and adds (Button - Add Point) points to the two data point arrays.
4.8 A change to the position of a point in the data point arrays must be user friendly. (Mouse click or drag in the Left or Right display area)
4.9 A widget must be provided so the user can choose between calculating the Fundamental Array or using the default values. (Check box)
4.10 The fundamental array calculation for the data point arrays should be saved to the same file as the data points.

5.0 Text to Fingerspell - The fsvis Application
An application must be developed that reads in and displays sequence of files that represent the URS.

5.1 The software must initialize a 1-D array with all of the files in the standardized fsvis format. (internal storage)
5.2 The software must initialize a 2-D array that cross references each 2-letter combination in the alphabet to the index position of the first file that represents that combination in the 1-D array, along with the number of files in the directory. (internal storage)
5.3 The software must parse the URS into a list of letter combinations that can be associated with the 2-D array. (internal storage)
5.4 The software must maintain a trace to all of the files and the order needed to display the URS as the string is being displayed (stack)
5.5 The software must provide a means to get an any-speed play back. (slider and the software can be displayed faster than real-time)

Appendix A. Original Proposal / Cross Reference to Requirements.

==> **trace** to implementation

Proposal: Fingerspell Video Library

Date: August 23, 2003
Number of Master of Science project hours: 6 hours

Language: C++

Interface: Routines that accept a string of text, a pointer to image data, update rate and rotation angle. The routine will load images into the image data buffer, at the requested rate, of hands performing sign language, in particular, fingerspelling.

==> **Project Objective**

Video Data Base: The data captured will be done in short clips of hands doing parts of words somewhat the same as a phoneme in speech. However, the clips will be of the most frequently occurring combinations of letters in words. For example, there will be an entry for, "in". The "in" might also be in another combination such as, "int". A method to determine which combination works best for a requested word will be developed.

The data captured will be either single jpegs, or, if possible compressed images into mpeg2 (m2v) files using software from the mpeg foundation. The jpeg, or mpeg, filename will reflect which letter(s) in the word is being signed. The name of the directory that the file is in will be the same as the sequence of letters that it holds.

==> **Approach Section II**

T**here will be two stages of data capture:**

1) **Single camera capture**

==> Single camera capture of entire 26x26 combination of letters. (Section II.3)

2) **3 camera capture**

The 3 camera capture will be converted to a single image using existing software, (from Leon Barrett, n-images->1 alignment). Once the conversion to a single image is made, the 3 camera capture will work the same as the single camera capture.

==> The 2 camera capture uses the morphing library from openCV (I.3,III.1).

==> single 2 camera capture of 1 fingerspelling position to demonstrate proof of concept. (See section III.3)

Software Requirements: The video library will have routines that take text input strings and find the video capture data that best represents the request. The software will concatenate the mpegs together to provide a seamless stream of data images to the image buffer. Or, if single jpeg images are stored, existing software will be used to get any-speed play back.

==> **see VI.2.2.3 fsvisG**

Program Environments: Three sample application environments will be provided. These will be the source and executables that link with the sign language video library. The following is a brief description of each environment.

1) Simple Storybook – A story will be defined in an external file with accompanying jpegs and sound files. The sign language video data that signs the text in the file, will appear at some location on the screen.
2) Computer/math lesson plan – A simple math/computer lesson play will be defined in an external file with accompanying jpegs. A FLTK GUI will be defined for excepting user inputs. Answers will be saved to a file.
3) Post-graduate psychology study – A questionnaire will be developed and results saved in a file or data base.

==> **Figure I.1, I.2, I.3**

Add ons:

Text to Speech: The text to speech will be done if existing free software can be found.

==> Peripheral Requirement not implemented

and

Face speaking the spelling of the words built the same way the fingerspelling is done.

==> Peripheral Requirement not implemented

Appendix B. How TO

This appendix provides a programmer with information useful in building the environment used in the Fingerspell Visualization project. The executables in the bin directory delivered with this project were built on rh9.0 because the target platform is a linux lab running red hat enterprise.

Appendix B. Table of Contents

==========

A. Downloads, used source rpms, programmer will have to find them.

* fltk[8] - transportable windowing program
* openCV[4] - open source computer vision
* kino[5] - process a video tape
* mjpegtools[7] - encodes jpegs into movie file format m2v.
* xine[6] - linux mpeg player

B. Login file Setup

The following variables were defined in the .bashrc file:

```
PROJECT_DIR=/data/wu
export PROJECT_DIR

# for wu Advance Graphics class
ARCHTYPE=linux
MESHTYPE=gmesh
DEBUGTYPE=debug
PROJHOME=${PROJECT_DIR}/assign
export ARCHTYPE
export MESHTYPE
export DEBUGTYPE
export PROJHOME
```

C. How to compile

All programs are compiled using the standard unix make function.
The software has been successfully run on both fedora core 5,(fc5)
and red hat 9.0, (rh9.0). However, openCV requirements have changed with both compiling and linking. The current version of the code compiles on both platforms. However, there is a change in library names. There are two versions of the directory:

${PROJHOME}/assign/make

The fedora core 5 make requirements are in the directory:

make_fc5

The red hat 9.0 version of make is in the directory:

make_rh9pt0

D. How to run applications
The file: csb_demo_17Apr07 is provided with the delivered source to run demo executables. The following describes how to run the executables.

* fsvisG - run program inside directory fs
    * Type in sentence with a through z and blanks.
    * converts upper case to lower case
    * Will not except any other characters,no periods,questionmarks...
    * Will not spell numbers
* fs2abc - run program inside directory fs
    ** Only does one word, no spaces
* iedG run with jpegs or pnk files in same directory or subdirectory
    ** viewing jpegs is the default program parameter: iedG
    ** to view all pnk files use program parameter: iedG pnk
    ** Modules:
        ** Black Out - Really changes to Green, the nosave color.

Will get position of mouse click and color, by one of following: Colors less than an RGB, Green all data with same color within a tolerance, Green all data to one side of mouse click

** Color Game - Change pixels to green when red < green, (or merge pnk file on top of back ground)

** Blur - Average

** Best Fit - Search for translation and rotation that best match two images

** Merge - Merge Out Same Pixels less than a tolerance,

** Merge a Background - loose all foreground green

** Morph - Morph to images using a given fundamental matrix. Has to have jpegs

** Many More: Rescale,Resize,Rotate,Translate,Translate to Location,Brush,Overlay

* bfitG bfit input_file output_file [pnk]

** run with jpegs or pnk files in same directory or subdirectory

** see main program, bfit.cpp

* morphPointsG

** only have images of same dimension in directory and sub directory

** read in point file from menubar (~.fund)

** read in image files from browser under image box (~.jpeg)

** calculation of fundamental matrix

** check "Use Calculation"

** hit button "Cal Fund Matrix"

** slider will show virtual image at different view angles

** "Save To Files" button saves to vm_1.pnk, ..._9.pnk

** can add new points, can't remove

** can move points using arrow buttons < >...

** Save, through menubar, saves points into fundamental matrix file.

E. Support Programs

* pnk2jpeg.cpp - read in file created by fsvisG to create a subdirectory

with pnk files named in format to subdirectory as jpegs, named in a sequential manner so that the files can be encoded to an mpeg[7] by the mpeg2 foundation software. The format of the command, once the files are jpegs is as follows on fedora core 5:

```
jpeg2yuv f 25 I p j fsvis_%04d.jpeg | mpeg2enc o ${1}.m2v
```

The command on red hat 9.0 is:

```
cat *.ppm | ppmtoy4m -F 25:1 | mpeg2enc -o ${1}.m
```

Where the command is inside an executable script file, run with the name of the movie file as an input parameter. The rh9.0 version uses ppm files. The following is an example of the unix command, convert, used to convert a file from jpeg to ppm:

convert *file_name*.jpeg *file_name*.ppm

The user must guarantee that the order of files listed by the first first part of the command is in the order needed for a movie.

* cpall.cpp- copy all files in one directory to another naming in the fingerspelling data base filename format.
* mv_vm.cpp move virtual images created by morphing process into fingerspelling database.

Appendix D. References to ASL in Education

1.) Susan Goodwyn, Linda Acredolo, and Catherine Brown (2000). Impact of symbolic gesturing on early language development. Journal of Nonverbal Behavior, 24 (2), pp. 81-103.
2.) Brie Moore, Linda Acredolo, & Susan Goodwyn (April 2001). Symbolic gesturing and joint attention: Partners in facilitating verbal development. Paper presented at the Biennial Meetings of the Society for Research in Child Development
3.) Linda Acredolo and Susan Goodwyn (1985). Symbolic gesturing in language development: A case study. Human Development, 28, 40-49.

Report References:

[1] "My First Baby Signs", HarperFestival by Acredolo, Goodwyn, and Gentieu
[2] Seitz,S.M., C.R.Dyer, C.R. VIew Morphing. Proc. SIGGRAPH 96, In Computer Graphics (1996),pp 21-30 ftp://ftp.cs.wisc.edu/computer-vision/papers.pages/subpages/seitz.1996.sigg/abstract.html
[3] "Liguorian" magazine, February 2004, p14, "Courtesy of DeafBlind UK, 100 Bridge St. Peterborough, PE11DY"
[4] www.opencv.com or http://sourceforge.net/projects/opencvlibrary/
[5] www.kinodv.org - Vision: "Easy and reliable DV editing for the Linux desktop with export to many usable formats."
[6] sourceforge.net/projects/xine
[7] sourceforge.net/projects/mjpeg
[8] www.fltk.org